Computer Architecture

Project 2

Due Date : 2022, 04, 21 23:55 PM

Project 2) Profiling MiBench with gem5

다음 지시사항에 맞춰 과제를 제출해 주시길 바랍니다.

1) 과제 스펙에서 요구하는 파일들을 모두 project2_학번.tar 파일 하나에 압축해 주세요.
   예) project2_2019314346.tar

2) 리눅스가 가상머신에 설치되어 있지 않으신 분들은 아래 파일을 사용하셔도 됩니다. (VMWare에서 사용할 수 있는 .ova 파일입니다.)

Ubuntu 16.04
https://drive.google.com/file/d/1SATXpRN8_iG_eSaXX77B1Mzb8Z1COv77/view?usp=sharing

Project 2
다음과 같은 환경에서 각 벤치마크를 gem5를 통해 실행하고, 각 실행의 stats.txt과 stdout.txt 파일을 제출하세요.

Configuration	config/example/se.py	
ISA	ARM	
CPU Type	[atomic / timing]	
Benchmarks	Binary	Input Set
	automotive/bitcount	3000
	security/sha	sha_50kbInput.asc 파일 (설명 참조)
	automotive/susan(corner)	input_small.pgm (설명 참조)

제출 파일 (압축 파일 안에 있어야 할 파일)

bitcount_atomic_stats.txt
bitcount_timing_stats.txt
bitcount_atomic_stdout.txt
bitcount_timing_stdout.txt

sha_atomic_stats.txt
sha_timing_stats.txt
sha_atomic_stdout.txt
sha_timing_stdout.txt

susan_corners_atomic_stats.txt
susan_corners_timing_stats.txt
susan_corners_atomic_stdout.txt
susan_corners_timing_stdout.txt
output_atomic_small.corners.pgm
output_timing_small.corners.pgm

※gem5의 설치 방법은
https://github.com/dependablecomputinglab/csi3102-tutorial
해당 페이지를 참조해 주시길 바랍니다. 중간에 문제가 생긴다면 이 파일의 끝에 있는 트러블슈팅을 참조해 주시길 바랍니다.

gem5는 기본적으로 다음과 같이 실행합니다.

./build/<isa>/<gem5binary> <config> -c <benchmark> 기타옵션들

이번 과제에선 ARM ISA를 사용하셔야 하며, cpu-type을 정해주셔야 합니다. gem5 내의 hello world를 예시로 하면 다음과 같습니다.

./build/ARM/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/arm/linux/hello --cpu-type=atomic
(--cpu-type=timing으로 실행하시면 timing cpu로 실행이 됩니다)

gem5를 실행할 때, 위처럼 실행하시면 커맨드라인을 입력했던 셸에 stats와 config를 제외한 모든 결과가 출력되게 됩니다.

이를 다른 파일에 출력되도록 나누려면 ./build/ARM/gem5.XXX 뒤에 다음과 같은 플래그들을 추가해 주시면 됩니다.
-re -d 폴더명 --stdout-file=stdout_파일명
--stderr-file=stderr_파일명 --stats-file=stats_파일명

mkdir my_dir
./build/ARM/gem5.opt -re -d my_dir --stdout-file=stdout.txt --stderr-file=stderr.txt --stats-file=stats.txt
configs/example/se.py -c tests/test-progs/hello/bin/arm/linux/hello --cpu-type=atomic

stdout: gem5 시뮬레이터의 standard output파일입니다. 기본적인 실행 정보들이 여기에 포함되어 있습니다. --output플래그가 없다면 벤치마크 자체적으로 printf등을 통해 출력하는 내용도 여기 포함됩니다.
stderr: gem5 시뮬레이터의 standard error파일입니다. 시뮬레이션 중 발생하는 Warning/Error가 여기에 저장됩니다.
stats: 시뮬레이션에 관한 다양한 정량적 정보들이 여기 포함됩니다. 예를 들면, 총 실행 시간, 총 실행한 연산 수, 캐시 힛/미스 수 등 다양한 정보들이 포함되어 있습니다.

벤치마크의 출력값(printf으로 나오는 값들)을 stdout에서 따로 빼고 싶으시다면 ./configs/example/se.py 뒤에
--output=output_파일명 를 추가해 주시면 됩니다. 단, “-d 폴더명“으로 지정한 폴더에 저장되지 않으므로 이를 직접 설정해주셔야 합니다.

./build/ARM/gem5.opt -re -d my_dir --stdout-file=stdout.txt --stderr-file=stderr.txt --stats-file=stats.txt configs/example/se.py --output=my_dir/output.txt -c tests/test-progs/hello/bin/arm/linux/hello --cpu-type=atomic

이렇게 해주시면, "Hello world!"만 my_dir/output.txt 파일에 저장됩니다.

※이번 과제에선 이를 분리하지 않은 채로 stdout을 제출하셔도 됩니다.

gem5 실행 예시를 tests/test-progs/hello/bin/arm/linux/hello 파일로 진행했습니다. 여러분들께서는 다음 페이지의 내용을 참조하셔서, miBench의 벤치마크들을 ARM에서 실행 가능하도록 빌드하시고 이를 gem5에서 실행하시면 됩니다.

예시)
mkdir ca_project2

./build/ARM/gem5.opt -re -d ca_project2 --stdout-file=bitcount_atomic_stdout.txt --stderr-file=bitcount_atomic_stderr.txt --stats-file=bitcount_atomic_stats.txt configs/example/se.py -c 직접_빌드한_bitcount_경로 --cpu-type=atomic

만약 프로그램이 입력값을 필요로 한다면 다음과 같이 -o를 사용해주셔야 합니다. (실행할 때 args 등으로 값을 넘겨줘야 할 경우)

./build/ARM/gem5.opt -re -d ca_project2 --stdout-file=bitcount_atomic_stdout.txt --stderr-file=bitcount_atomic_stderr.txt --stats-file=bitcount_atomic_stats.txt configs/example/se.py -c 직접_빌드한_bitcount_경로 --cpu-type=atomic -o "3000"

※ MiBench 설치 및 크로스 컴파일러를 통한 빌드 방법은
https://github.com/dependablecomputinglab/csi3102-gem5-profiling
해당 페이지를 참조해 주시길 바랍니다.

설명을 덧붙이자면, 각각의 벤치마크 경로로 가시면 Makefile이 있습니다.
susan의 경우를 예로 들면, Makefile을 여시면

```
susan: susan.c Makefile
	gcc -static -O4 -o susan susan.c -lm
clean:
	rm -rf susan output*
```

위와 같은 내용을 확인할 수 있습니다.

크로스 컴파일을 하시려면, Makefile을 아래와 같이 수정해 준 뒤,

```
susan: susan.c Makefile
	arm-linux-gnueabi-gcc -static -O4 -o 바이너리_이름 susan.c -lm -static
clean:
	rm -rf susan output*
```

(바이너리 이름은 자유롭게 하시면 됩니다. 예) susan_ARM)
위와 같이 변경한 뒤 커맨드 라인에서 make를 실행해 주시면 됩니다.

과제 2에서 실행하실 벤치마크들은, 모두 위와 같은 간단한 방법으로 빌드가 가능합니다.
굳이 Makefile을 수정하고 싶지 않으시다면, 위에서 변경한
arm-linux-gnueabi-gcc -static -O4 -o 바이너리_이름 susan.c -lm -static
명령어를 커맨드라인에 쳐 주셔도 괜찮습니다.

※마찬가지로, 각 벤치마크 폴더에 runme_**.sh 파일이 있습니다. 해당 파일은 기본 Makefile로 빌드된 벤치마크들을 실행하는 스크립트를 포함하고 있습니다.**

우선 bitcount를 예로 들겠습니다.
runme_small.sh를 여시면
bitcount의 runme_small.txt를 보시면

```
#!/bin/sh
bitcnts 75000 > output_small.txt
```

bitcount는 이와 같이 실행합니다. 이번 과제에선 실행시간 단축을 위해 gem5에 실행 옵션을 3000 으로 변경해서 실행하셔야 합니다. 따라서,

gem5를 실행하실 때 -o “3000” 을 붙여주셔야 합니다. (> output_small.txt는 리눅스에서 자체적으로 출력을 해당 파일로 보내라는 것을 의미합니다. gem5로 치면 --output=에 해당합니다.)

sha의 경우

```
#!/bin/sh
sha input_small.asc > output_small.txt
```

이렇게 돼있을 겁니다. 이번 과제에선 실행시간 단축을 위해 "sha_50kbInput.asc"파일을 함께 첨부해 드리니 -o "sha_50kbInput.asc파일이있는위치/sha_50kbInput.asc“로 실행해주시면 됩니다.

susan의 경우

```
#!/bin/sh
susan input_small.pgm output_small.smoothing.pgm -s
susan input_small.pgm output_small.corners.pgm -e
susan input_small.pgm output_small.corners.pgm -c
```

이렇게 입력파일 결과파일 function(smoothing/corners/corners)으로 실행하며, 이번 과제에선 corners만 하시면 됩니다.
보시면 3번째 파라미터는 output파일을 의미합니다. 따라서 gem5실행 시 -o 옵션을

timing CPU에 대해선 -o “input_small이_있는_경로/input_small.pgm output을_저장할_경로/output_timing_small.corners.pgm -c”
atomic CPU에 대해선 -o “input_small이_있는_경로/input_small.pgm output을_저장할_경로/output_atomic_small.corners.pgm -c”

위와 같이 주시면 됩니다 (뒤 설명 참조). input_small.pgm은 susan벤치마크가 있는 폴더 내에 있습니다. susan의 경우 --output=옵션으로 출력되는 결과 (printf 등으로 출력되는 텍스트)가 아닌, output_timing(atomic)_small.corners.pgm 파일이 실제 output에 해당합니다.

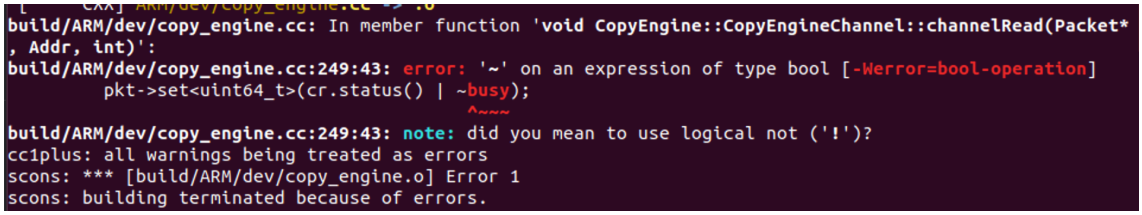
※gem5 실행에서 -o로 파일을 넘길 때 “~/ 를 사용하시면 대부분 실행이 잘 되지 않습니다. 절대 경로를 사용하신다면 ~/를 /home/사용자이름/ 으로 바꿔서 실행해 주세요.

트러블슈팅

1. 가상머신을 쓰신다면 메모리 크기를 3~4GB 이상 잡아주시는 것을 권장합니다.
만약 메모리 크기가 부족한데 늘릴 수 없다면, 다음과 같이 swap 메모리를 설정하셔서 진행하실 수 있습니다. swap 메모리는 일종의 가상메모리이며, 시스템 메모리가 부족할 경우 사용하게 됩니다.

sudo fallocate -l <원하는 가상메모리 크기, ex) 2G> /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile

2. 저희가 gem5를 오래 전의 stable로 빌드하시도록 요청드렸는데 (git checkout 2f3c467) 우분투 18.04 버전 이상을 쓰고 계신 분들은 gcc및 g++가 7 이상이 버전이라 충돌이 발생합니다.



이 경우, 다음과 같이 해주시면 됩니다.

1) gcc-5 및 g++-5 설치
sudo apt-get install gcc-5
sudo apt-get install g++-5

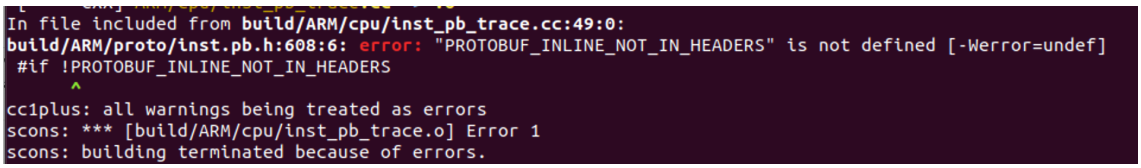
2) 버전 확인
gcc -v (7.XX가 뜰겁니다)
gcc-5 -v (5.5.0 혹은 5.4.0이 뜰겁니다)
g++ -v (7.XX가 뜰겁니다)
g++-5 -v (5.5.0 혹은 5.4.0이 뜰겁니다)

3) scons 커맨드 변경
scons build/ARM/gem5.opt
이 커맨드라인 대신에

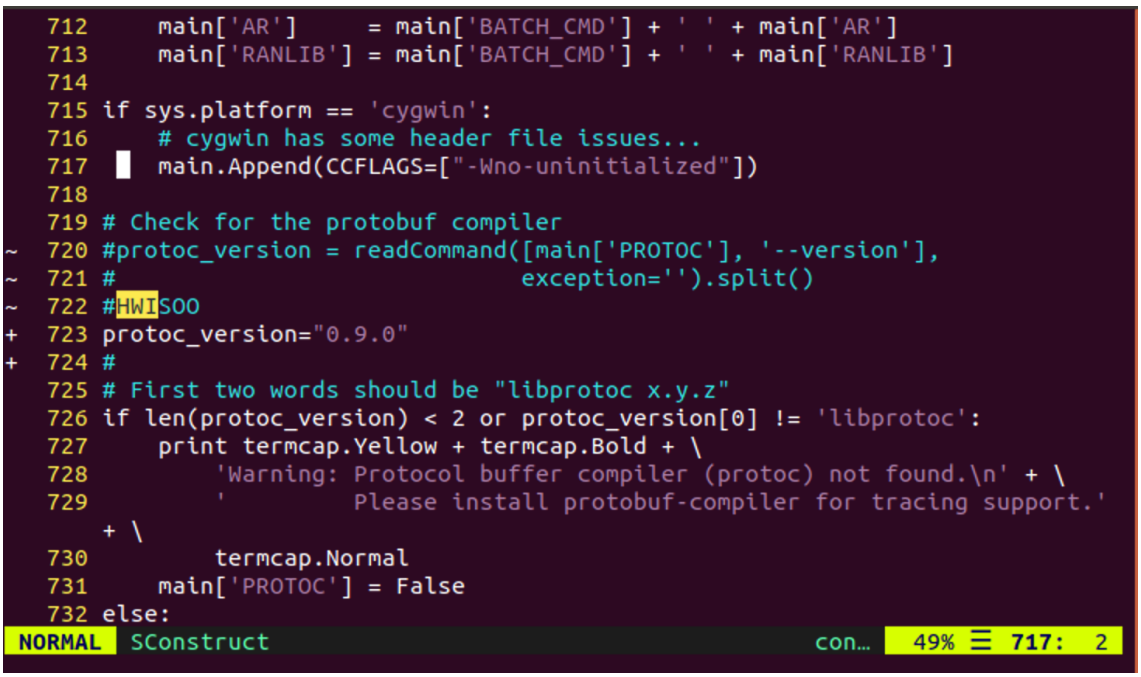
scons CC=gcc-5 CXX=g++-5 build/ARM/gem5.opt
이렇게 실행해주세요.

4) 같은 에러가 반복된다면 build폴더를 지우고 다시 시도해주세요

3. libprotoc 버전이 gem5와 충돌되는 에러입니다.



gem5에서 libprotoc이 필수는 아니지만 (일부 기능이 안돌아가는데 저희 과제완 관련이 없습니다) 최신버전이 설치되어있으면 충돌이 발생합니다.
충돌이 발생할 경우 gem5 폴더의 SConstruct 파일을 다음과 같이 수정해주세요.



사진에 보이시는 것처럼 (#HWISOO 참조)
protoc_version = readCommand([main['PROTOC'], '--version'], exception='').split() (사진의 720번 줄)을 주석 처리해주시고
protoc_version = "0.9.0" (사진의 723번 줄)을 입력해주시면
gem5를 빌드할 때 protoc의 버전 요구 조건 이하임을 확인하고 protoc 없이 컴파일을 진행합니다.

4. 혹시 gem5 빌드가 완료된 후 miBench 어플리케이션을 돌릴 때 kernel too old 에러가 발생하시는 분은
src/arch/arm/linux/process.cc에서

```
63  static SyscallReturn
64  unameFunc32(SyscallDesc *desc, int callnum, Process *process,
65              ThreadContext *tc)
66  {
67      int index = 0;
68      TypedBufferArg<Linux::utsname> name(process->getSyscallArg(tc, index));
69
70      strcpy(name->sysname, "Linux");
71      strcpy(name->nodename, "m5.eecs.umich.edu");
72      //strcpy(name->release, "3.0.0");
73      strcpy(name->release, "4.4.0+");
74      strcpy(name->version, "#1 Mon Aug 18 11:32:15 EDT 2003");
75      strcpy(name->machine, "armv7l");
76
77      name.copyOut(tc->getMemProxy());
78      return 0;
79  }
80
81  /// Target uname() handler.
82  static SyscallReturn
83  unameFunc64(SyscallDesc *desc, int callnum, Process *process,
84              ThreadContext *tc)
85  {
86      int index = 0;
87      TypedBufferArg<Linux::utsname> name(process->getSyscallArg(tc, index));
88
89      strcpy(name->sysname, "Linux");
90      strcpy(name->nodename, "gem5");
91      //strcpy(name->release, "3.7.0+");
92      strcpy(name->release, "4.4.0+");
93      strcpy(name->version, "#1 SMP Sat Dec  1 00:00:00 GMT 2012");
94      strcpy(name->machine, "armv8l");
95
96      name.copyOut(tc->getMemProxy());
97      return 0;
98  }
```

빨간 박스처럼 주석친 부분을 바로 아래처럼 수정해주신 뒤, scons를 다시 돌려주세요.

(한 번 빌드하신 후라면 수정된 부분과 이를 사용하는 부분들만 컴파일이 돼서, 처음만큼 오래 걸리지는 않습니다)

5. fatal: syscall openat (#322) unimplemented.

gem5에 있는 hello world는 실행이 되는데, 직접 크로스컴파일 한 바이너리를 돌리면 이렇게 뜨시는 분들이 계실겁니다.
해당 에러가 뜨는 것은 arm-linux-gnueabi-gcc의 버전이 마찬가지로 7 버전인데, 저희가 사용하는 gem5의 버전이 이와 호환이 되지 않기 때문입니다.

다음과 같이 진행해주시면 됩니다.

1) sudo apt-get install gcc-5-arm-linux-gnueabi
이렇게 하시면 arm-linux-gnueabi-5 가 설치됩니다.

2) Makefile의 gcc를 arm-linux-gnueabi-gcc에서 arm-linux-gnueabi-gcc-5로 수정한 뒤 make를 다시 해주세요

5.1 5번으로도 322에러가 해결되지 않으신다면

src/arch/arm/linux/process.cc에서

/* 322 */ SyscallDesc("openat", unimplementedFunc),

이 부분을 찾아

/* 322 */ SyscallDesc("openat", openatFunc<ArmLinux32>),

이렇게 수정해주신 뒤 scons를 다시 실행해주세요.